



**THE UNITED STATES PATENT AND TRADEMARK OFFICE**

**Appl. No.** : 09/848,713  
**Applicant** : Doug Grumann  
**Filed** : May 3, 2001  
**Title** : METHOD AND APPARATUS TO EXTRACT THE HEALTH OF  
A SERVICE FROM A HOST MACHINE  
**TC/A.U.** : 2442  
**Examiner** : Benjamin A. Ailes  
**Docket No.** : 10002681-1  
**Customer No.** : 022879

**Mail Stop Appeal Brief - Patents**  
Commissioner of Patents  
P.O. Box 1450  
Alexandria, Virginia 22313-1450

**APPEAL BRIEF UNDER 37 C.F.R. §41.37**

## **TABLE OF CONTENTS**

	<b><u>Page</u></b>
<b>I. REAL PARTY IN INTEREST.....</b>	<b>3</b>
<b>II. RELATED APPEALS AND INTERFERENCES .....</b>	<b>4</b>
<b>III. STATUS OF CLAIMS.....</b>	<b>5</b>
<b>IV. STATUS OF AMENDMENTS.....</b>	<b>6</b>
<b>V. SUMMARY OF CLAIMED SUBJECT MATTER.....</b>	<b>7</b>
<b>VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL .....</b>	<b>9</b>
<b>VII. ARGUMENT .....</b>	<b>10</b>
<b>VIII. CLAIMS APPENDIX.....</b>	<b>i</b>
<b>IX. EVIDENCE APPENDIX.....</b>	<b>vi</b>
<b>X. RELATED PROCEEDINGS APPENDIX.....</b>	<b>vii</b>

**I. REAL PARTY IN INTEREST**

Hewlett Packard Company is the real party in interest.

**II. RELATED APPEALS AND INTERFERENCES**

There are no other related appeals or interferences.

### **III. STATUS OF CLAIMS**

Claims 1, 2, 4 - 8, 11, 12, 14 - 22 and 24 are pending. Claims 1, 2, 4 - 8, 11, 12, 14 - 22 and 24 are rejected. Claims 3, 9, 10, 13, and 23 are cancelled. Applicant appeals the rejection of claims 1, 2, 4 - 8, 11, 12, 14 - 22 and 24.

**IV. STATUS OF AMENDMENTS**

There are no amendments filed after final.

## **V. SUMMARY OF CLAIMED SUBJECT MATTER**

Applicants have discovered a new, improved method, and a corresponding mechanism, for determining the service health of a computer system. The method includes gathering performance information and translating the gathered performance information, into generic health metrics. The result is an abstracted set of consistent service health metrics that can be provided to the performance monitoring tools such that the performance monitoring tools may use the health metrics without needing to know how the health metrics were derived. This method decouples the performance tool implementation from the metric derivation and removes dependencies between the services, their implementation, and the management tool set. For example, a tool may use the generated service level violation metric to generate an alert when violations raise above a threshold. The performance monitoring tools do not need to know anything about the service, its instrumentation, or how the service level metric is calculated. The tool simply compares the resultant metric against its threshold. The performance monitoring tool uses a programmatic interface library call or script interface to access health metrics for all current services. If the underlying application changes, the current version of the performance monitoring tool is unaffected because of this consistent interface. As a result, the system administrator does not necessarily need to install a new version of the performance monitoring tool. Thus, the apparatus and method are extensible without propagating a dependency up into the higher levels of the management software.

Referring to Figure 3 and its accompanying description starting at page 9, line 20, a system 100 includes a health generator 10 that provides an output of service health data 14 to performance monitoring tools 13. The service health metrics may be directly measured or derived from applications, processes and thread instrumentation, for example. The method is independent of specific provider applications and management tool sets, thereby allowing for shorter time-to-market for service management solutions. The output 14 of the method may be either in the form of a programmatic or scriptable interface 108 to be used by the monitoring tools 13, which are capable of reporting status of many disparate computer services. The tools 13 may reside on different systems and architectures and may be supplied by different vendors. To accommodate different performance monitoring tools, the interfaces are generic and flexible.

Considering the invention recited in claim 1, and referring to Figures 3 and 6, a method 200 for dynamically determining the health of a service resident on a host machine includes collecting 203 service performance information 12 from the resident service,

wherein the collected service information relates to a plurality of performance metrics; and translating 207 the collected service performance information into a generic output 14 relating to current operational performance of the service, wherein the generic output 14 is one of a scriptable interface and an application programming interface 108, and useable by different performance monitoring tools 13. See page 9, line 20 to page 10, line 24.

Considering the invention recited in claim 11, and referring to Figures 3 and 4, a health generator 10, which is used to determine a health of a service resident on a host machine, includes a data collection engine 121 that collects service health information 12 and a translation data analysis engine 123 that translates the collected service health information into a generic health output 14 in the form of an application programming interface or a scriptable interface 108 and useable by performance monitoring tools 13. See page 9, line 20 to page 11, line 16.

Considering the invention recited in claim 18, and referring to Figures 3 and 6, a method 200 for monitoring health data of a service operating on a host machine includes collecting 203 service performance information from the service; collecting 203 external performance information from components of the host machine; translating 207 the collected service and external performance information according to a health generation algorithm to generate a generic service health output 14; and providing 209 the generic service health output 14 relating to current operational performance of the service as an output file accessible by performance monitoring tools, wherein the generic service health output is one of a scriptable interface and an application programming interface 108, and usable by the different performance monitoring tools 13. See page 9, line 20 to page 10, line 24 and page 20 line 20 to page 15, line 21.

Considering the invention recited in claim 21, and referring to Figures 3 and 4, an apparatus 10 determines a health of a service, wherein the service operates on a host computer 100, the apparatus 10 including a collection module 121 that receives performance information related to the service; a translation health generator module 125 that applies a rule set 127 to the received performance information and derives generic health metrics 14 therefrom; and an output module 125 that outputs the generic health metrics relating to current operational performance of the service, wherein the generic health metrics are in a format usable by different performance monitoring tools 13. See page 9, line 20 to page 11, line 16 and page 12, line 19 to page 14, line 19.



## **VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL**

Claims 1, 2, 4 - 6, 8, 11, 12, 14, 17 - 22 and 24 are rejected under 35 U.S.C. § 103(a) over “Systems Management: Application Response Measurement (ARM) API” (hereafter ARM API) in view of U.S. Patent 6,633,908 to Leymann, et al. (hereafter Leymann).

Claims 7 and 15 are rejected under 35 U.S.C. § 103(a) over ARM API in view of U.S. Patent 5,949,976 to Chappelle (hereafter Chappelle).

Claim 16 is rejected under 35 U.S.C. § 103(a) over ARM API in view of U.S. Patent 6,647,413 to Walrand et al. (hereafter Walrand).

More specifically, and considering independent claims 1, 11, 18, and 21, the Examiner asserts, *inter alia*, that ARM API teaches all that is recited except that ARM API does not “explicitly recite the translation of the performance information into a generic output.” However, the Office Action then asserts that Leymann “teaches these features” and more specifically that Leymann teaches use of a generic output “wherein Leymann teaches in column 8, lines 3 - 14 the making of data available for all applications requesting the data by use of an invocation agent. The invocation agent provides the data needed by calling applications and is considered a generic component due to its independence from specific applications. Because of the independence of the invocation agent, the data it provides is therefore considered to be ‘generic output’ as required by the claims. The output provided by the invocation agent is deemed generic because it is made available for all applications.” The Examiner contends that one skilled in the art would have been motivated to combine ARM API and Leymann to “specifically implement independence from a specific application and make the data available for a wide range of calling applications as taught by Leymann in column 8, lines 9 - 13.

## VII. ARGUMENT

### CLAIM 1

#### ARM API

ARM API is directed to a method for measuring service levels of applications in which a number of performance measurement agents collect response time information. *See* ARM API pages 16 - 17. ARM is part of an “open technical standard,” with its own unique components, systems, and techniques. As a preliminary matter, ARM API is designed to measure “performance time” and is not intended for “transaction monitoring.” *See* ARM API, page 6. To enable the agents to collect this “performance time” information, each application must first be modified to include code that invokes the various performance functions. *See* ARM API page 5. The thus-modified applications include calls to the ARM API, where the calls are placed by an application writer so as to define the start and end of a business transaction. Properly denoting these start and end points a key task of the application writer and ensure that accurate service level data are obtained by the ARM API compliant measurement agents. ARM API does not disclose or suggest use of a generic output.

#### Leymann

Similar to ARM API, Leymann is directed to performance monitoring of applications. In fact, Leymann starts with the ARM API system and then proposes some modification, none of which read on or render obvious the inventions recited in the instant application. Specifically, Leymann departs from ARM API by not requiring modification to each application so as to invoke system calls. Instead of instrumenting applications as in ARM API, Leymann instruments invocation agents 202 and uses these invocation agents to collect the desired performance information. *See* column 8, lines 1 - 23. The invocation agent’s sole function is to call a data monitoring function (the ARM) to first start data collection and then terminate data collection. To do this, Leymann takes the instrumentation that in ARM API would be appended to a specific application and appends that same instrumentation to the invocation agents 202. Thus, Leymann’s invocation agents 202 are hardly generic. Instead, Leymann describes a system in which the ARM API’s function calls (i.e., what is appended to the ARM API applications - collection start and stop points) is migrated to the measurement or collection agent (in Leymann’s parlance, the invocation agent 202). “The present invention allows to enable the management and measurement of application performance in systems management without special instrumentation of the corresponding applications.” *See* Leymann, column 7, lines 62 - 65. However, “[t]he present invention

contemplates instrumenting the invocation agents instead ...” See column 7, line 67 - column 8, line 1. Leymann asserts that instrumenting the invocation agents has certain advantages over instrumenting the applications. But in either ARM API or Leymann, some specific instrumentation has to take place to ensure compatibility with the ARM standard because either the application (as in ARM API) or the invocation agent 202 (as in Leymann) has to make “the appropriate ARM calls”. See column 8 lines 7 - 8. Leymann refers to the invocation agents 202 as “generic components” but goes on to explain that they are generic only in the sense that they are “independent from the specific application.” See column 8, lines 9 - 11. Furthermore, the output of Leymann’s “generic” invocation agent 202 hardly is “generic.” As Leymann specifically points out, the output must be specific to the ARM API - which certainly is not a generic output. Thus, Leymann specifically invokes use of the ARM standard, and because Leymann’s system outputs data specific to the ARM standard, Leymann does not disclose or suggest use of a generic output. Instead, Leymann’s output is ARM-specific. In short, all that Leymann has done is move the non-generic API calls from an application to an agent. The agent, in “a first step, requests the ARM to measure the response of the application ... [i]n a second step the invocation agent invokes the application instance, and in a third step the invocation agent requests the ARM to terminate the response measurement.” See column 4, lines 59 - 65. Thus Leymann’s output is not generic.

#### **The Invention of Claim 1**

In contrast to ARM API and Leymann, the invention recited in claim 1 says nothing about instrumenting applications or instrumenting monitoring agents. Claim 1 does not recite a generic monitoring agent as in Leymann. Claim 1 does not recite instrumenting applications as in ARM API. Claim 1 puts no restrictions on the monitoring agent as in Leymann. Instead, and in contrast to Leymann and ARM API, the invention recited in claim 1 is a method for dynamically determining the health of a service including the step of “translating the collected service performance information into a generic output relating to current operational performance of the service.” Leymann uses a generic monitoring agent. Claim 1 recites translating the collected performance information into a generic output.

More specifically, claim 1 recites a method for determining the service health of a computer system. The method includes gathering performance information and translating the gathered performance information, into a “generic output.” The “generic output” is an abstracted set of consistent service health metrics that can be provided to the performance monitoring tools such that the performance monitoring tools may use the health metrics without needing to know how the health metrics were derived. This method decouples the

performance tool implementation from the metric derivation and removes dependencies between the services, their implementation, and the management tool set. For example, a tool may use the generated service level violation metric to generate an alert when violations raise above a threshold. The performance monitoring tools do not need to know anything about the service, its instrumentation, or how the service level metric is calculated. The tool simply compares the resultant metric against its threshold. The performance monitoring tool uses a programmatic interface library call or script interface to access health metrics for all current services. If the underlying application changes, the current version of the performance monitoring tool is unaffected because of this consistent interface. As a result, the system administrator does not necessarily need to install a new version of the performance monitoring tool.

The feature of the “generic output” is defined in the specification, for example, at page 5, lines 3 - 20:

However the performance information is gathered, the apparatus and method translate the gathered performance information, or metrics, into health metrics. The result is an abstracted set of consistent service health metrics that can be provided to the performance monitoring tools such that the tools may use these metrics without needing to know how the health metrics were derived. This decouples the performance tool implementation from the metric derivation and removes dependencies between the services, their implementation, and the management tool set. For example, a tool may use the generated service level violation metric to generate an alert when violations raise above a threshold. The performance monitoring tools do not need to know anything about the service, its instrumentation, or how the service level metric is calculated. The tool simply compares the resultant metric against its threshold. The performance monitoring tool uses a programmatic interface library call or script interface to access health metrics for all current services. If the underlying application changes, the current version of the performance monitoring tool is unaffected because of this consistent interface. As a result, the system administrator does not necessarily need to install a new version of the performance monitoring tool. Thus, the apparatus and method are extensible without propagating a dependency up into the higher levels of the management software.

Also note page 7, lines 12 - 23:

To solve these problems, a method and an apparatus are used to derive consistent service health measures by combining various instrumentation from both internal sources and external sources that relate to the service under observation. The service health metrics may be directly measured or derived from the applications, processes and thread instrumentation, for example. The method is independent of specific provider applications and management tool sets, thereby allowing for shorter time-to-market for service management solutions.

The output of the method may be either in the form of a programmatic or scriptable interface to be used by high-level monitoring tools that are capable of reporting status

of many disparate computer services. The tools may reside on different systems and architectures and may be supplied by different vendors. To accommodate different performance monitoring tools, the interfaces are generic and flexible.

Finally, note page 13, lines 15 - 17:

The rules set 127 provides algorithms and rules to translate the metrics supplied by the data collection engine 121 into a generic format that is usable by the performance monitoring tools.

As the above-quoted passages from the specification demonstrate, the term “generic output” is an interface that allows the health metrics to be used without any need to know how the health metrics were derived, and that removes any dependencies between the services, their implementation, and the management tool set. This is very different from what ARM API and Leymann disclose or suggest.

Leymann and claim 1 both use the word “generic.” But, any person skilled in the art will appreciate that the use of generic in claim 1 and the use of generic in Leymann are for very different purposes. Again, Leymann discloses a generic invocation agent that is coded to make specific ARM API calls. Leymann’s system does not, however, result in a generic output; instead, the output is ARM-specific. Leymann never discloses translating the ARM-specific output into a generic output. The invention of claim 1 recites a generic output.

Because Leymann and claim 1 use completely different generic features, Leymann cannot be used in combination with ARM API to render the subject matter of claim 1 obvious.

For all the reasons discussed above, claim 1 is patentable over ARM API in view of Leymann.

#### **CLAIM 11**

Independent apparatus claim 11 recites a data analysis engine that translates the collected service health information ... and provides one or more generic health metrics, wherein the generic output comprises a plurality of service health metrics, and wherein the translation data analysis engine combines one or more of the plurality of performance metrics to provide one or more of the plurality of service health metrics, and wherein the plurality of service health metrics comprises availability, capacity, throughput, service time, queue length, utilization, service level violations, and user satisfaction. As noted above, ARM API and Leymann, individually and in combination, do not disclose or suggest all of these features. Accordingly, claim 11 also is patentable.

### **CLAIM 18**

Independent method claim 18 recites the step of generating a generic service health output, wherein the generic output comprises a plurality of service health metrics, and wherein the translation data analysis engine combines one or more of the plurality of performance metrics to provide one or more of the plurality of service health metrics, and wherein the plurality of service health metrics comprises availability, capacity, throughput, service time, queue length, utilization, service level violations, and user satisfaction. For the same reasons that claims 1 and 11 are patentable, claim 18 is also patentable.

### **CLAIM 21**

Independent apparatus claim 21 recites service health metrics comprising availability, capacity, throughput, service time, queue length, utilization, service level violations, and user satisfaction. As noted above with respect to claim 1, ARM API and Leymann, individually and in combination, do not disclose or suggest all of these features. Accordingly, claim 21 is also patentable.

### **DEPENDENT CLAIMS**

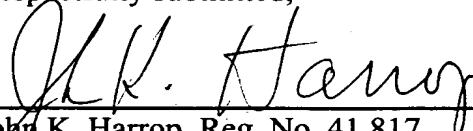
Claims 2 and 4 - 8 depend from patentable claim 1; claims 12 and 14 - 17 depend from patentable claim 11; claims 19 and 20 depend from patentable claim 18; and claims 22 and 23 depend from patentable claim 21. For these reasons and the additional features they recite, claims 2, 4 - 8, 12, 14 - 17, 19, 20, 22, and 24 also are patentable.

In view of the above amendments and remarks, Applicants respectfully submit that the application is in condition for allowance and request withdrawal of the rejection of the claims under 35 U.S.C. § 103(a).

The appeal brief fee in the amount of **\$510.00** is submitted herewith. However, should there be any additional fees required for this appeal brief, please charge any fees required or credit any over payment to **Deposit Account 08-2025** pursuant to 37 C.F.R. § 1.25.

Date: **June 19, 2009**

Respectfully submitted,

  
\_\_\_\_\_  
John K. Harrop, Reg. No. 41,817  
**ANDREWS KURTH LLP**  
1350 I Street, NW  
Suite 1100  
Washington, D.C. 20005  
Telephone: (202) 662-3050  
Fax: (202) 662-2739

## **VIII. CLAIMS APPENDIX**

1. (currently amended): A method for dynamically determining the health of a service resident on a host machine, comprising:

collecting service performance information from the resident service, wherein the collected service information relates to a plurality of performance metrics; and

translating the collected service performance information into a generic output relating to current operational performance of the service, wherein the generic output is one of a scriptable interface and an application programming interface, and useable by different performance monitoring tools,

wherein the generic output comprises a plurality of service health metrics, and wherein the translating step comprises combining one or more of the plurality of performance metrics to provide one or more of the plurality of service health metrics, and wherein the plurality of service health metrics comprises availability, capacity, throughput, service time, queue length, utilization, service level violations, and user satisfaction.

2. (previously presented): The method of claim 1, wherein the host machine comprises one or more components, further comprising:

collecting external performance information from one or more of the one or more components;

translating the collected external performance information; and

combining the translated external performance information and the translated service performance information to provide the generic output.

3. (cancelled):

4. (original): The method of claim 1, further comprising accessing the generic output to read the health of the service.

5. (original): The method of claim 1, wherein the collecting step comprises reading performance information provided by the service.



6. (original): The method of claim 1, wherein the collecting step comprises deriving performance information from the service.
7. (original): The method of claim 6, wherein the deriving step comprises using a wrapper program to read the performance information.
8. (original): The method of claim 6, wherein the deriving step comprises using a probe program to read the performance information.
9. (cancelled):
10. (cancelled):
11. (previously presented): An apparatus that determines a health of a service resident on a host machine, comprising:
  - a data collection engine that collects service health information; and
  - a translation data analysis engine that translates the collected service health information using a health generation algorithm and provides a generic output comprising one or more generic health metrics relating to current operational performance of the service, wherein the generic output is one of a scriptable interface and an application programming interface, and useable by different performance monitoring tools, wherein the collected service health information relates to a plurality of performance metrics, wherein the generic output comprises a plurality of service health metrics, and wherein the translation data analysis engine combines one or more of the plurality of performance metrics to provide one or more of the plurality of service health metrics, and wherein the plurality of service health metrics comprises availability, capacity, throughput, service time, queue length, utilization, service level violations, and user satisfaction.
12. (previously presented): The apparatus of claim 11, wherein the host machine comprises one or more external components, wherein the data collection engine collects external performance information from one or more of the one or more external components information, and wherein the translation data analysis engine translates the collected external information using the health generation algorithm to provide the one or more generic health metrics.

13. (cancelled):

14. (previously presented): The apparatus of claim 11, wherein the data collection engine, comprises:

a data query module that reads performance information from the service; and  
a data derivation module that derives performance information from the service.

15. (original): The apparatus of claim 14, wherein the data derivation module derives the performance information from one or more of a wrapper program, a benchmark program and a probe program.

16. (original): The apparatus of claim 11, wherein the health generation algorithm comprises:

a weighting scheme that weights one or more performance information parameters;  
a summation scheme that combines one or more performance information parameters;  
and  
a averaging scheme that averages collected service health information for a service health metric.

17. (original): The apparatus of claim 11, further comprising an interval control engine that receives the service health information at a first time interval and provides an output having a second time interval different from the first time interval.

18. (previously presented): A method for monitoring health data of a service operating on a host machine, comprising:

collecting service performance information from the service;  
collecting external performance information from components of the host machine;  
translating the collected service and external performance information according to a health generation algorithm to generate a generic service health output; and  
providing the generic service health output relating to current operational performance of the service as an output file accessible by performance monitoring tools, wherein the generic service health output is one of a scriptable interface and an application programming interface, and usable by the different performance monitoring tools, wherein the collected service information relates to a plurality of performance metrics, wherein the generic output

comprises a plurality of service health metrics, and wherein the translating step comprises combining one or more of the plurality of performance metrics to provide one or more of the plurality of service health metrics, and wherein the plurality of service health metrics comprises availability, capacity, throughput, service time, queue length, utilization, service level violations, and user satisfaction.

19. (original): The method of claim 18, wherein the step of collecting the service performance information comprises reading first service performance parameters and deriving second service performance parameters, and wherein the step of collecting the external performance information comprises reading first external performance parameters and deriving second external performance parameters.

20. (previously presented): The method of claim 18, further comprising collecting the service performance information on a first time interval and adjusting the first time interval to provide the generic service health output at a second time interval.

21. (previously presented): An apparatus that determines a health of a service, wherein the service operates on a host computer, comprising:

a collection module that receives performance information related to the service;

a translation health generator module that applies a rule set to the received performance information and derives generic health metrics therefrom; and

an output module that outputs the generic health metrics relating to current operational performance of the service, wherein the generic health metrics are in a format usable by different performance monitoring tools, wherein the generic health metrics comprise availability, capacity, throughput, service time, queue length, utilization, service level violations, and user satisfaction.

22. (original): The apparatus of claim 21, wherein the collection module receives external performance information from one or more external services coupled to the host computer and receives internal performance information related to operation of the service on the host computer.

23. (cancelled):

24. (previously presented): The apparatus of claim 21, wherein the generic health metrics is one of a scriptable interface and an application programming interface.

## **IX. EVIDENCE APPENDIX**

No evidence submitted.

**X. RELATED PROCEEDINGS APPENDIX**

No related proceedings.